

Table Compression in Oracle9i Release 2: A Performance Analysis

An Oracle White Paper
January 2003

Table Compression in Oracle9i Release 2: A Performance Analysis

Introduction.....	3
How Table Compression Works.....	3
Test Schema Configuration (Star and NF)	4
Space savings of Compression.....	5
Star Schema.....	7
Normalized Schema (TPC-H/ TPC-R)	8
Why the Star Schema Compresses Better Than the normalized TPC-H Schema	8
Performance Analysis.....	10
Star Query Examples.....	11
Star Query 1.....	11
Star Query 2.....	13
Star Query 3.....	13
Discussion of Star Query Performance	13
TPC-H Queries.....	16
TPC-H Query 1	16
TPC-H Query 6	16
TPC-H Query 15	16
Discussion TPC-H Query Performance	17
Best Practices.....	18
Conclusion.....	18
References.....	19
Appendix – SQL queries	20

Table Compression in Oracle9i Release 2: A Performance Analysis

INTRODUCTION

Table Compression, a new feature introduced in Oracle9i Release2, can be used to compress entire tables, table partitions and materialized views. It will drastically reduce the disk space and buffer cache requirements and in many cases improve query performance especially on IO bound systems. Compression targets decision support and OLAP applications, but other areas might also experience great benefits. After explaining how table compression works the paper introduces two types of schemas that are commonly used to implement decision support systems (both OLAP and data warehouses), namely a star schema and a normalized schema. Using these schemas the two main sections of this paper analyze how table compression can result in tremendous space saving and investigate the impact of table compression on queries.

HOW TABLE COMPRESSION WORKS

Oracle9i Release2 compresses data by eliminating duplicate values in a database block. The algorithm used is a lossless dictionary-based compression technique. Compressed data stored in a database block is self-contained. That is, all the information needed to recreate the uncompressed data in a block is available within that block. The algorithm decides, based on column length and number of occurrences, whether to create an entry in the symbol table (dictionary) for a particular column. Only entire columns or sequences of columns are compressed. For short values and values with few occurrences no symbol table entry is created. All occurrences of this value are replaced with a short reference to the symbol table. In order to achieve optimal compression results, columns might be reordered within one block. However, this is transparent to the user.

In comparison to other data compression techniques where a fixed number of symbol table entries, typically 256, are used to compress an entire table, Oracle's data compression implementation has many benefits. Firstly, in Oracle the number of symbol table entries are chosen by the system on a block basis during load time to yield optimal compression results. Secondly, the symbol table entries are created by the system and do not need to be tuned by the user. Thirdly, Oracle's algorithm dynamically adapts to changes in data distribution without compromis-

Invoice_ID	Cust_Name	Cust_Addr	Sales_amt
1233033	Meyer	11 Homestead Rd	13.99
1212300	Meyer	11 Homestead Rd	1.99
1243012	Meyer	11 Homestead Rd	1.99
9923032	McGryen	3 Main Street	1.99
9833023	McGryen	3 Main Street	1.99
2133056	McGryen	3 Main Street	1.99

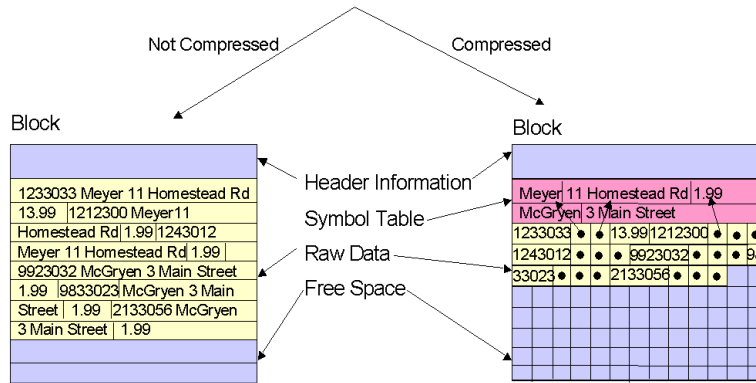


Figure 1: Compressed Block vs. not Compressed Block

ing the compression factor. Consequently, to create a compressed table one only needs to include the keyword `COMPRESS` into the table definition.

Figure 1 illustrates the differences between storing data in a compressed versus uncompressed block. With the exception of a symbol table at the beginning, compressed database blocks look very much like regular database blocks. Code modifications done in the Oracle RDBMS server to allow for compression were very localized. Only the portions of the code dealing with formatting the block, and accessing rows and columns were modified. As a result, accessing a compressed block is completely transparent to the database user or any application, and all database features and functions that work on regular database blocks also work on compressed database blocks.

TEST SCHEMA CONFIGURATION (STAR AND NF)

There are a wide variety of ways of arranging schema objects in the schema models designed for data warehousing. One data warehouse schema model is a star schema. Another schema is the third normal form (3NF) schema. Additionally, some data warehouse schemas are neither star schemas nor 3NF schemas, but instead share characteristics of both schemas; these are referred to as hybrid schema models.

The Oracle9i database is designed to support all data warehouse schemas in the most efficient way. Some features may be specific to one schema model (such as the star transformation feature, which is specific to star schemas). However, the vast majority of Oracle's data warehousing features are equally applicable to star schemas, 3NF schemas, and hybrid schemas. Key data warehousing capabilities

such as partitioning (including the rolling window load technique), parallelism, materialized views, and analytic SQL are implemented in all schema models.

The determination of which schema model should be used for a data warehouse should be based upon the requirements and preferences of the data warehouse project team. Comparing the merits of the alternative schema models is outside of the scope of this paper. Instead, this paper uses examples of a star schema and a normalized schema to illustrate the benefits of compression for both of these schemas.

Figure 2 describes the star schema example emphasizing its typical structure of star where the fact table DAILY_SALES is the center surrounded by the dimensions

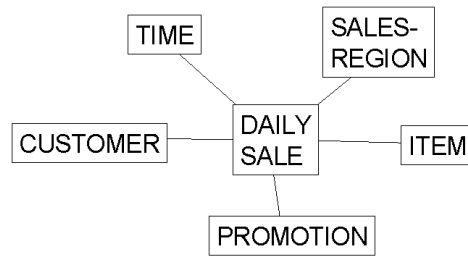


Figure 2: Typical Star Schema

TIME, CUSTOMER, SALES REGION, ITEM and PROMOTION. There are two summary tables defined on DAILY_SALE: WEEKLY_SALES and WEEKLY_AGGR. WEEKLY_SALES aggregates DAILY_SALES for each item

and customer to weekly numbers. WEEKLY_AGGR builds on DAILY_SALES by aggregating further on sales region.

In our second example we demonstrate compression on a different type of setup, namely that of the standard decision support benchmarks TPC-H/TPC-R. Its schema consists of eight base tables modeling the data warehouse of a typical retail

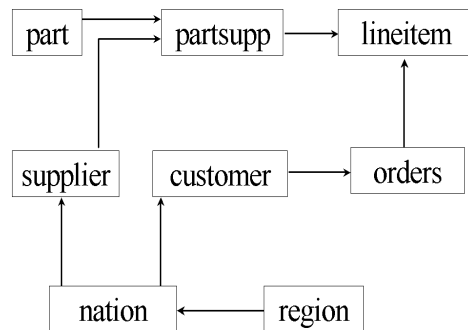


Figure 3: TPC-H and TPC-R Schemas

environment (see Figure 3). Tables such as PART, SUPPLIER, PARTSUPP and CUSTOMER contain relatively static information about the items a typical retail company buys from their supplier and sells to their customer. These tables amount to about 15% of the total database. The two largest tables, LINEITEM and ORDERS contain about 85% of the total database size. They contain numerical

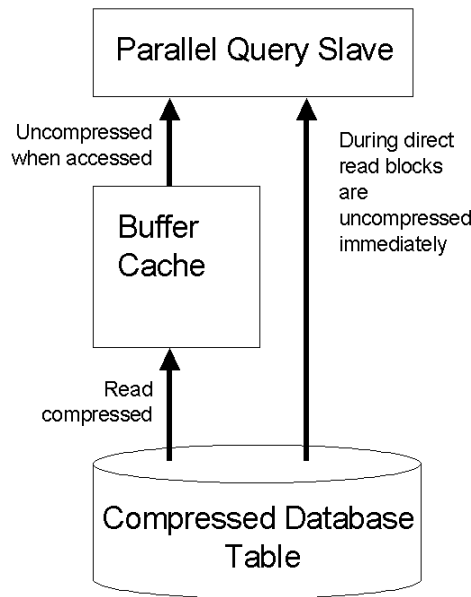
measurements similar to a fact table in our star schema example. Contrary to our previous example this schema is not organized as a star schema but follows a normalized [3] approach.

SPACE SAVINGS OF COMPRESSION

Table compression can significantly reduce disk and buffer cache requirements for database tables. Since the compression algorithm utilizes data redundancy to com-

compress data at a block level, the higher the data redundancy is within one block, the larger the benefits of compression are. Although there might be data redundancy across data blocks, that data cannot be used to further compress data.

If a table is defined “compressed” it will use fewer data blocks on disk, thereby, reducing disk space requirements. Data from a compressed table is read and cached in its compressed format and it is decompressed only at data access time. Because data is cached in its compressed form, significantly more data can fit into the same amount of buffer cache (see Figure 4).



Before quantifying the space savings of compression using the two example schemas we define compression factor and space savings: Compression factor (CF) of a table is defined as the ratio between the number of blocks required to store the non-compressed object compared to the number of blocks needed for the compressed version:

$$CF = \frac{\#non_compressed_blocks}{\#compressed_blocks}$$

Figure 4: Data Access Path with Compression

The space savings (SS) are therefore defined as:

$$SS = \frac{\#non_compressed_blocks - \#compressed_blocks}{\#non_compressed_blocks} \times 100$$

The compression factor mostly depends on the data content at the block level. Unique fields, (fields with a high cardinality) such as primary keys cannot be compressed, whereas fields with a very low cardinality can be compressed very well. On the other hand, longer fields yield a larger compression factor since the space saving is larger than for shorter fields. Additionally, if a sequence of columns contains the same content, the compression algorithm combines those columns into a multi column compression element for even better compression results.

The symbol table, which holds the values of fields with multiple occurrences, is automatically generated by Oracle for each block. In most cases, larger block sizes increase the compression factor for a database table as more column values can be linked to the same symbol table.

Sorting data before loading can further increase the compression factor. The more fields of the same content that are concentrated in each block the more efficiently the compression algorithm works. If one knows that one or multiple fields of a database object have similar values - indicated by a low number of unique values - sorting the data on those fields is likely to increase the compression factor. However, sorting on fields with very low cardinality does not necessarily yield a large compression factor increase. Due to the low cardinality of this field, rows with the same value can be found already at a high concentration in each block. Therefore, best results can be achieved by sorting on a field that is both long and has a medium cardinality.

Star Schema

Fact and summary tables are usually the largest tables in a star schema representing 70% or even more of the total database size. In contrast dimension tables are very small. Hence, compressing dimension tables does not yield an overall large disk savings and should only be considered when dealing with very large dimensions. We therefore only compress fact tables and materialized views for our test schema configuration.

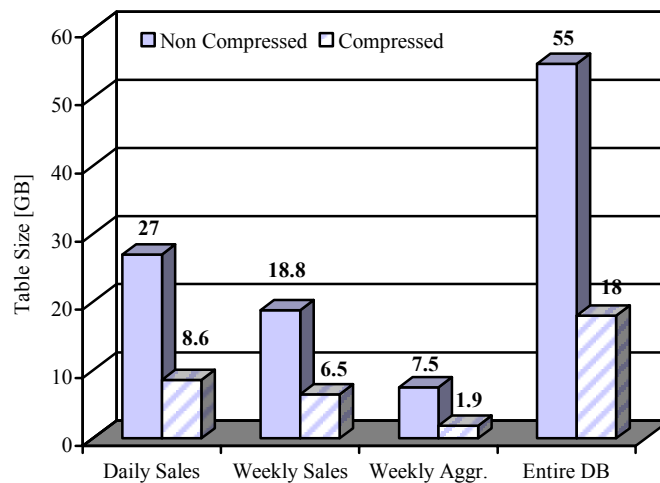


Figure 5: Compression Factors for the Star Schema

Figure 5 illustrates how well data of the star schema compresses. The size for DAILY_SALES decreases from 27 GB to 8.6 yielding a compression factor of 3.1. The two materialized views compress at compression factors of 2.9 and 4.0. WEEKLY_SALES shrinks from 18.8 GB to 6.5 while WEEKLY_AGGR shrinks from 7.5 GB to 1.9 GB yielding a space savings of 67 to 75 percent. That is, the compressed version of WEEKLY_SALES requires only 25% disk and buffer cache space than their uncompressed counterpart while the compressed versions of DAILY_SALES and WEEKLY_SALES/WEEKLY_AGGR require only 33% of the resources that their uncompressed counterparts use. The overall database size reduced from 55 GB to 18 GB. The space saving, only compressing the fact tables

and their materialized views, achieved on the customer's entire star schema is about 67% at a compression factor of about 3.1.

Normalized Schema (TPC-H/ TPC-R)

The normalized schemas for TPC-H and TPC-R are dominated by the two tables `LINEITEM` and `ORDERS`. These tables store numerical measurements similar to the fact table in the star schema containing about 75% of all data. Compression for `LINEITEM` is the highest at a compression factor of about 1.6, while `ORDER` compresses at a compression factor of about 1.2 (see Figure 6). This means that the compressed versions of `LINEITEM` and `ORDERS` consume only about 60% to 80% of the uncompressed tables. The overall compression factor for the TPC-H database is about 1.4 resulting in a space saving of about 29%.

Figure 6 shows the table sizes for `LINEITEM` and `ORDERS` in compressed and non-compressed modes. `LINEITEM` shrinks from 79 GB to 49 GB while `ORDERS` shrinks from 17 GB to 14 GB resulting in compression factors of 1.6 and 1.2 respectively. The overall database sizes is reduced from 115 GB to 82 GB (CF=1.5).

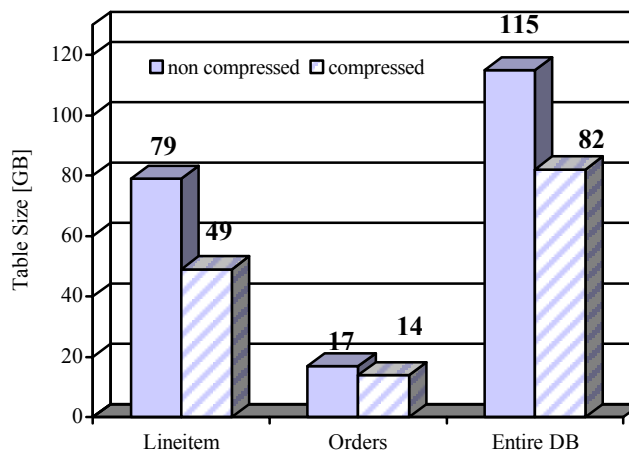


Figure 6: Compression Factors for the Normalized Schema

WHY THE STAR SCHEMA COMPRESSES BETTER THAN THE NORMALIZED TPC-H SCHEMA

In the previous two sections we have discussed the compression factors and space savings that can be obtained on a real world star schema and the normalized TPC benchmark schema. Using table compression on all fact and summary tables, the compression factor for the star schema is about 3.1, resulting in a space savings of about 67%. That is, the database footprint on disk shrinks to less than half than without compression. On the other hand, the normalized schema using compression on the two largest tables, which are equivalent to fact tables in the data warehouse, yield only a compression factor of 1.4 and a space savings of 24%.

The compression factor of a table depends on the redundancy of its data. Data is compressed by eliminating duplicate values in each database block. Higher data redundancy yields generally to a higher compression factor. A table that contains many columns with low number of distinct values, as indicated by the DBA_TAB_COL_STATISTICS dictionary table, does not automatically yield a high compression factor. It rather depends on the data distribution and the average column length of each individual column. Obviously, the data distribution determines the number of potential distinct values, and the average column length determines the number of records stored in one block.

Schema	Column	Total number of rows in Million	Total number Distinct values	Avg. number Rows per Block	Calculated number Distinct Values per Block	Measured avg. number Distinct Values per Block
DAILY-SALES (Star)	address	142	49901	146	146	2
	region id	142	41	146	41	10
LINE ITEM (TPC)	comment	600	3528002	46	46	45
	discount	600	11	46	11	11

Figure 7: Compression in Star and TPC-H/R Tables

A close look at the number of distinct values in two representative columns (ADDRESS, REGION_ID) of the star schema table DAILY_SALES and two columns (COMMENT, DISCOUNT) of the TPC-H/R schemas table LINEITEM explains why data of our star schema example compresses better than data of the TPC-H/TPC-R databases. For each of these columns Figure 7 displays the total number of rows, the total number of distinct values, the average number of rows per block¹, the calculated number of distinct values in each block and the measured average number of distinct values per block². When calculating the number of distinct values per block we assume a uniform distribution of rows. That is, the calculated number of distinct values per block equals the average number of rows per block if the total number of distinct values is larger than the number of rows per block, otherwise it equals to the total number of distinct values. Consider the following, uniformly distributed sequence of the numbers {1,2,...5}, which represent a one-column row with the number as its content:

1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Let's further assume that the average number of rows per block is 4. Then the row to block mapping would look like this:

1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
Block1				Block2				Block3				Block4				Block5			

¹ Taken from the DBA_TAB_COL_STATISTIC dictionary table

² Representative sample blocks were investigated.

As one can see each block has 4 distinct values. However, this sequence has been chosen to demonstrate the effect of uniformly distributed data rather than being a general case. The reader can verify that a different sequence can result in fewer distinct values per blocks.

For the star schema columns the measured average number of distinct values is significantly lower than the calculated number of distinct values. It indicates that the data are not uniformly distributed, but are clustered together. This is very common for data warehouse fact and summary tables; in almost every Data Warehouse environment periodical data maintenance applies some kind of grouping or sorting to the new data. For instance, an ETL process, consolidating new fact table information from sources, has to compare and aggregate, thus sort, the various sources prior to the insertion into the fact table. Similar data clustering occurs naturally in summary tables that perform group by or advanced OLAP operations such as rollup and cube.

On the other hand, in the TPC-H and TPC-R schemas the actual number of distinct values in a block is the same as the calculated number of distinct values per block indicating a strict uniform distribution of its data. As a matter of fact the data generator of TPC-H/R has been criticized for generating normally distributed data.

If data is not clustered together, like in the TPC-H and TPC-R case, sorting the data before loading can significantly increase compression. Which columns one should include in a sort depends on their cardinality and average length. Generally, a long column yields a larger increase in compression compared to a short column. As for the cardinality, it turns out that sorting on very low cardinality columns such as GENDER or MARITAL STATUS is less effective as sorting on medium cardinality columns. The optimal columns to sort on seem to be those that have a table-wide cardinality equal to the number of rows per block. Sorting columns with a lower cardinality than the number of rows per block is less efficient because column values already occur in a high redundancy. Sorting on columns with any higher cardinality column also results in a soaring compression increase.

PERFORMANCE ANALYSIS

This Section investigates the performance impact of compressed objects on query performance. Queries might gain significant performance if their data is stored in compressed tables. As described in the section about space savings, the number of blocks required to store data in compressed tables can be considerably less than in non-compressed tables. Thus, queries against compressed tables read fewer database blocks, directly improving query read performance.

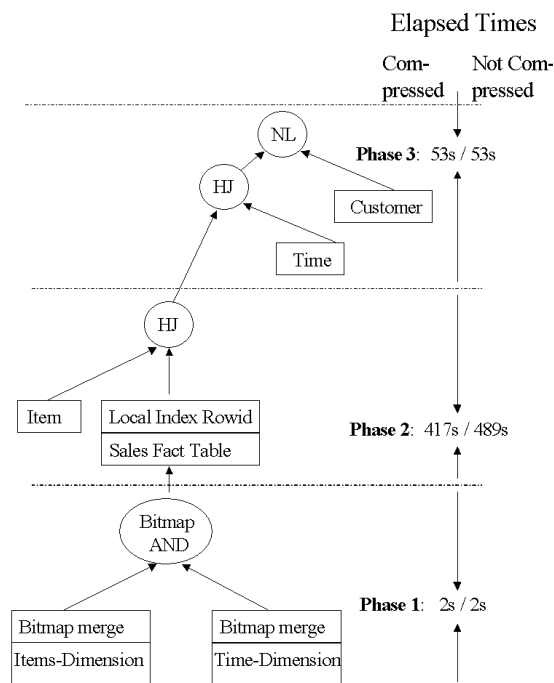
In order to show the benefits of compression on queries against a star schema we analyze the performance of 3 star queries, which were also drawn from the customer environment of the star schema example in the previous section. To demonstrate the benefits of compression on a normalized schema we analyze TPC-H

queries from a recently published TPC-H 100 GB result [1]. The full benchmark description is available on the TPC website (www.tpc.org). Please note, the elapsed times for the compressed runs are taken from the published TPC-H benchmark, while the non-compressed times are obtained during the benchmark tuning phase. TPC-H is comprised of a set of 22 business queries designed to exercise system functionalities in a manner representative of complex business analysis applications. Discussing all 22 queries would be far beyond the scope of this paper. Therefore, we will limit our discussion to a representative subset (Query 1, 6 and 15).

Star Query Examples

Queries against a star schema use Oracle's star transformation feature. The star transformation is a powerful optimization technique for star queries that relies upon implicitly rewriting SQL of the original star query. Oracle's cost-based optimizer automatically chooses the star transformation whenever appropriate. The star transformation is a cost-based query transformation aimed at executing star queries efficiently. Oracle processes a star query using three phases. In the first phase Oracle accesses all bitmap indexes on dimensions for which the query defines predicates. The resulting bitmap vectors are then concatenated with set operators (AND/OR) depending on the query's logic. This step includes the conversion of the final bitmap vector to row identifiers. The second phase retrieves exactly the necessary rows from the fact table utilizing the row identifier set of the previous phase and joins it to the first dimension. The third phase joins this result set to the dimension tables to retrieve detailed data necessary to complete the query. It is important to mention that the end user never needs to know any of the details about the star transformation.

Star Query 1



The first query calculates the final sales sum of specific items for the years 1998 and 1999 for specific months and customer. The results are grouped by year, months, district type and item. The complete SQL statement can be found in the appendix; Figure 8 shows its execution plan and the time spent for the different operation steps, both with the uncompressed

Figure 8: Execution Plan and Time Spent in Different Query Phases for Star Query 1

and compressed fact table SALES.

Oracle's Optimizer recognizes that this query is a star query and transforms it as discussed above. The fact table is accessed through a bitmap access path based on a bitmap AND, of the two merged bitmaps TIME and ITEM. Based on those bitmaps Oracle can efficiently calculate all qualifying rowids of the DAILY_SALES table. Since the fact table contains only references to the dimension tables but not their detailed data, the dimensions need to be joined back to the fact table. This is done in the subsequent join operations on ITEM and TIME. Since the select list contains detailed data from the CUSTOMER table we also need to join to the CUSTOMER dimension. The execution plan, furthermore, shows the time spent for the certain operation steps using a compressed/uncompressed fact table respectively.

Star Query 2

The second query calculates the sum of sales made from all customer in Chicago whose deliveries are in specific sales regions for the first five months in 1998 and 1999. The result set is grouped by customer name, district name, sales region, year and month.

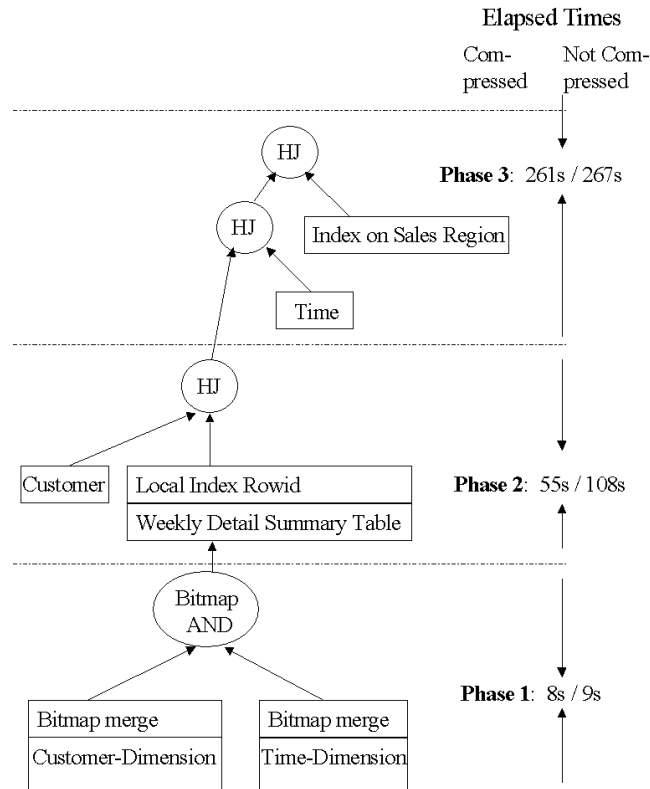


Figure 9: Execution Plan for Star Query 2

In contrast to Query 1 this query uses the materialized view WEEKLY_SALES which contains sales data rolled up to weeks. The SQL can be found in the appendix. The query plan and the time spent for certain operation are shown in Figure 9.

This query is also transformed leveraging the star transformation feature in Oracle's optimizer.

Star Query 3

The third query is a variant of Query 2. Instead of calculating the sum of sales made in the first six month of 1998 and 1999 it calculates the sale of the entire two years 1998 and 1999. The query plan remains the same as that of Query 2 and is not explicitly shown.

Discussion of Star Query Performance

We execute the above queries against a non-compressed schema and subsequently against a compressed schema. Figure 9 illustrates the elapsed times and performance speedup³ for the three star queries for both runs. Each pair of bars shows the elapsed times for one query. The first bar indicates the elapsed time of this query against the non-compressed schema, while the second bar indicates the elapsed

³
$$ElapsedTimeSpeedup = \frac{elapsedTime_{non_compressed} - elapsedTime_{compressed}}{elapsedTime_{non_compressed}}$$

time against the compressed schema. The speedup is shown as a percent number for each pair of bars. Query 1 shows an elapsed time speed up of 13%, Query 2 one of 15% and Query 3 one of 11%. The overall elapsed time speed up for all 17 customer queries is 16.5%.

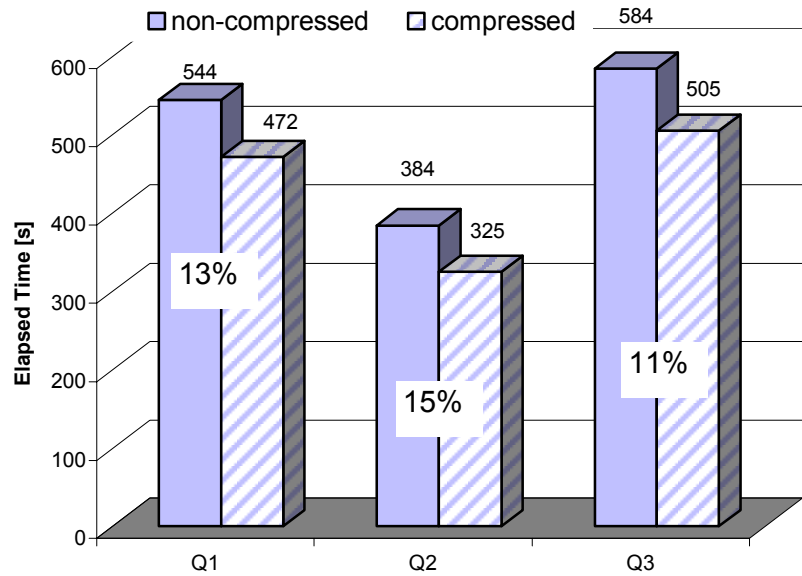


Figure 10: Query Elapsed Time Saving Using Compressed Tables

The elapsed time speedup of star queries executed against a compressed star schema results from a speedup in accessing the smaller fact or summary tables during the second phase of query execution. The larger the compression factor the greater is the elapsed time speedup. However, one cannot derive a query elapsed time speedup from the compression factor because the speedup depends on how many blocks and how sparse the blocks are that need to be accessed by the query. Furthermore, the query speedup depends on how IO bound the system is. The weaker the IO subsystem, the larger is the compression benefit.

Q1	Compression	Phase 1	Phase 2	Phase 3
1	no	2	489	53
	yes	2	417	53
2	no	8	108	267
	yes	9	55	261
3	no	12	162	405
	yes	14	85	419

Figure 11: Detailed Query Timings for Different Phases of Star Queries

In Query 1 the first Phase comprises of accessing two bitmap indexes (on CUSTOMER and TIME dimensions) and concatenating their results with an AND operator. The total elapsed time for this phase is about two seconds regardless of the schema because this operation does not access any compressed tables. The second phase joins the DAILY_SALES Fact Table to the CUSTOMER dimension utilizing a hash join. Most of the time in this phase is spent scanning the probe table (DAILY_SALES) because the build table (CUSTOMER) is very small. The third phase joins the results set of the previous phase with three dimensions ITEM, TIME and CUSTOMER. Similarly to the first phase, this phase is about 53 seconds regardless of compression. The small differences between compressed and non-compressed query runs in Phase 1 and Phase 3 are in the noise level. With table compression this query takes about 472 seconds while without it takes about 544 seconds, a decrease in elapsed time of 72 seconds or 13%.

The only step involving compression, which results in different runtime behavior, in this query is the access of the fact table DAILY_SALES. For Query 2 and Query 3 it is the access of the summary table WEEKLY_SALES. Figure 11 lists the detailed times for accessing the bitmap indexes (Phase 1), the first join back (Phase 2), which accesses the fact/summary tables and the remaining join backs to dimensions of all three queries (Phase 2).

Only the first join-back (join to fact/summary table) consistently shows a significant elapsed time difference when executed against a compressed table. For instance, in Query 2 this join has an elapsed time difference of about 53 seconds (17%) while performing the join backs to the other dimensions show no significant elapsed time differences. Being the probe table for this hash-join the fact table is accessed using rowids of the previous phase. Elapsed time improvement of this hash join using table compression is limited by the dominance of the probe part. In general, an operation that has a large IO time benefits more from compression than an operation that has a small IO time.

At the first look the small query elapsed time improvement of 11% seems to be inconsistent with the large fact table compression ratio of 2.9 (=67% space savings). Taking into consideration that the second phase accounts for about 90% of the query execution time, the 67% space savings should reduce the elapsed time by roughly 60%. This is true if the rows that are accessed consolidate into consecutive blocks. For instance if an operation accesses 5 rows in 5 different blocks, spread widely across a non-compressed table, regardless of the compression factor, those 5 rows won't likely fall into the same block if the table is compressed. Consequently, even for the compressed table, the query will read 5 blocks. This is why operations that do not exhibit a locality of reference property benefit less from compression. This can be proven by counting the number of blocks needed to be read from the compressed and the non-compressed table. Accessing the DAILY_SALES fact table Query 1 reads about 92115 blocks in the compressed case and 94137 blocks in the non-compressed, a difference of only 2022 blocks or about 8.15%. This indicates that rows accessed by Query 1 are widely spread

across the fact table resulting in no locality of reference property and, therefore, benefits less from compression (Query 2 and Query 3 observe similar behavior).

TPC-H Queries

This section briefly describes the three TPC-H queries whose performance we will analyze in subsequent sections.

TPC-H Query 1

Query 1 performs multiple aggregations and summaries by reading and processing over 95% of the rows of the database's largest table. Only this single table is scanned with a very small result set. Since this query is performing many aggregations it is typically CPU bound.

TPC-H Query 6

Query 6 accesses the large detail table only (Lineitem) selecting about 12% of the rows, and returning a single column answer. It performs very few aggregations making it a very good candidate for stressing the IO subsystem.

TPC-H Query 15

In this table, 1/28th of the Lineitems (3 months) are selected based on date and aggregated on Supplier. Close to all the Suppliers in the database will participate as output from this first step aggregation. The query itself returns details from the Supplier row which represents the maximum revenue. One row is returned from this query.

Discussion TPC-H Query Performance

Figure 12 shows the elapsed times and speedups achieved with compression of the three TPC-H queries 1, 6, and 15. This Figure is organized similarly to Figure 11. It shows that Query 1 has a relatively small slowdown of about 2 percent, which might well be in the measurement error range, when run against the compressed

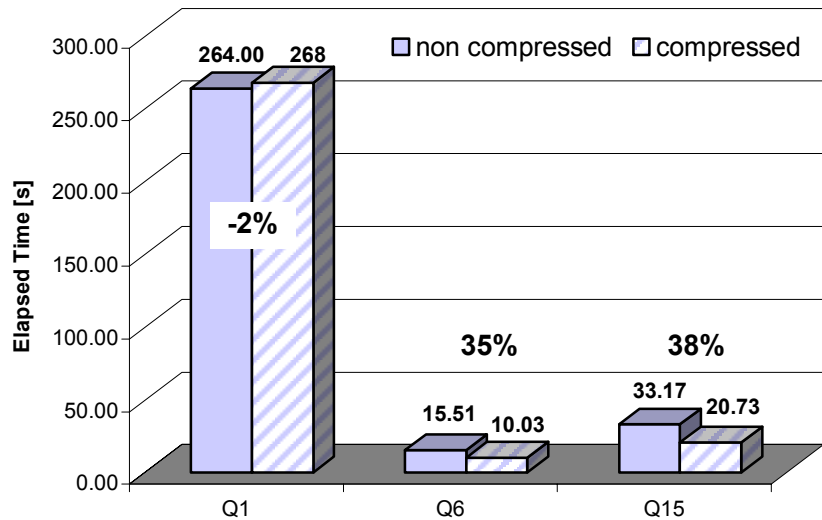


table `LINEITEM`. On the other hand Query 6 and Query 15 show a significant elapsed time speedup. Query 6 shows a speedup of 35% while Query 15 shows a 38% speedup. The overall speedup⁴ of all 22 TPC-H queries run against the compressed database and against the non-compressed database is about 10%. Elapsed time for the insert test (rf1) increased by about 3.9% while the elapsed time for the delete test (rf2) decreased by about 17%. The primary metric for TPC-H, `QphH@100GB`, improved by about 10%. For some of the queries we observed a slight CPU overhead. As demonstrated in the section about Space Savings, TPC-H data yield a compression factor of only 1.2 for `ORDERS` and 1.6 for `LINEITEM`. Queries against the compressed database perform on average 27% fewer disk accesses.

Elapsed Time of Query 1 increases by about 2%. This can be explained by the slight increase in CPU utilization and the fact that this query is CPU bound. The total CPU consumption in the compressed case increases by about 2%. Since there is no CPU left for query execution, elapsed time increases by about the same amount as the total CPU consumption increases. The large decrease in disk utiliza-

⁴
$$OverallSpeedup = \sum_{i=1}^{i=22} \frac{ElapsedTimeNonCompressedQi - ElapsedTimeCompressedQi}{ElapsedTimeNonCompressedQi}$$

tion of about 38% has very little impact on this query because the disk subsystem is not the bottleneck during this query.

Query 6 performance improves by about 35%. This query being IO bound leaves much of the available CPUs unused. Consequently, the increase of CPU utilization can be easily compensated for by the system without degrading performance.

Similar to Query 6, Query 15 benefits from table compression showing a 38% performance increase. This query performs multiple join operations. Similarly to query 6, this query leaves some of the available CPU unused decreasing query elapsed time by about 8%. This query benefits greatly from compression reading about 42% less data in case of the compressed database.

Using compressed tables in TPC-H reduces the elapsed time for most queries in our system setup. But increases the elapsed time for others. IO bound queries directly benefit from compression because it reduces the average consumption of the bottlenecking resource, namely the disk subsystem. CPU bound queries also benefit from fewer disk subsystem consumption. However, the overhead in accessing compressed tables can hurt elapsed time of these queries.

BEST PRACTICES

Table compression is best used for read intensive workloads predominant in business intelligence applications. Compression is recommended for large tables, such as fact tables and materialized views of star schemas and tables of 3NF schemas that contain transactional data. It is not recommended for small tables such as dimensions of star schemas because of their insignificant contribution to the overall space savings.

For partitioned tables, it is possible to compress some or all partitions. For instance in a large data warehouses where data is usually partitioned, not frequently accessed partitions can be compressed maximizing space utilization. If a partition is still being modified by an ETL process, the overhead of compression should be evaluated.

In order to increase the compression factor, sorting data of a table before compressing it can increase the compression factor. However, as discussed earlier, in many cases business intelligence data is naturally clustered and, therefore, yield very good compression factors without additional effort. Further tips on how to use Oracle new compression feature can be found on the Oracle Technology Network web site (<http://otn.oracle.com>).

Furthermore, in most cases, larger block sizes increase the compression factor for a database table as more column values can be linked to the same symbol table.

CONCLUSION

The cost of disk sub-systems can be a very large portion of building and maintaining large data warehouses. Oracle9i Release2 helps reduce this cost by compressing

the data stored in an Oracle database, and it does so without the typical trade-offs of space savings versus access time to data.

The compression factor that can be achieved depends on the data and its distribution. Our test cases, discussed in this paper, show compression factors between 1.2 and 4.0 (17% to 67% space savings). However, higher compression ratios have been observed. For example, detailed call data from a major telecom company resulted in a compression factor of 12 (92% space savings). Among the customer test results, this was the highest compression ratio achieved. A compression factor of 5 (80% space savings) was achieved on aggregated sales data from different customers in different industries.

Analyzing queries against a star schema and a 3NF schema we have shown that table compression can result in great query performance increase. This is mostly attributed to the decrease in disk IO and the very inexpensive decompression operation needed to access compressed table data.

Execution of queries against star and 3NF schemas is substantially different. In star queries most of the execution time is spent in accessing bitmap indexes and then joining qualifying rows of the fact table to dimension tables. Accessing bitmap indexes is therefore not affected by table compression. On the contrary, queries against a 3NF schema perform extensive hash join operations between possibly large portions of the database. These operations benefit greatly since less data is read from disk.

Queries of our star schema example show improvements of about 12-16 % yielding an overall performance increase in all 17 queries of about 16.5 %. On the other hand performance improvements of TPC-H queries reach 38%. The overall improvement of all TPC-H queries reaches 10%.

Table compression in Oracle 9i Release 2 significantly reduces disk space and buffer cache requirements resulting in improved. These benefits are obtained without requiring any application changes.

REFERENCES

- [1] TPC-H 100 GB published 07/15/02 by HP/Oracle on Alpha Server ES45 and Oracle 9iR2 Executive Summary: http://www.tpc.org/results/individual_results/HP/es45_5578_es.pdf FDR http://www.tpc.org/results/FDR/tpch/es45_5578_fdr.pdf
- [2] Oracle9i Data Warehousing Guide Release 2 (9.2) Part Number A96520-01
- [3] Date C.J., "An Introduction to Database Systems", Reading, Mass., Addison Wesley Verlag, 1981

APPENDIX – SQL QUERIES

Star Query 1:

```
SELECT T.year, T.month, C.district, I.name,
       SUM(sales)
FROM   customers C,
       daily_sales S,
       items I,
       time T
WHERE  S.item_nr=I.item_nr
AND    S.addr_id=C.addr_id
AND    S.date=T.date
AND    T.year in (1998, 1999)
AND    T.month in (1-05-1998, 1-06-1998, ...
                  ..., 1-03-1999)
AND    C.district in ('CO', 'CA')
AND    I.group = 'classic'
AND    I.item = 'blue gravave'
GROUP BY T.year, T.month, C.district, I.name;
```

Star Query 2:

```
SELECT C.district, C.name,
       T.year, T.month,
       R.region_number,
       SUM(sales)
FROM   customers C,
       time T
       weekly_sales S,
       sales_region R
WHERE  S.region_id=R.region_id
AND    S.addr_id=C.addr_id
AND    S.date=T.date
AND    T.year in (1998, 1999)
AND    T.month in (1,2,3,4,5)
AND    C.district = 'Chicago'
AND    R.region_number in ('234', '4565', '111', '1')
GROUP BY
       C.district, C.name, T.year, T.month,
       R.region_number;
```



Table Compression in Oracle9i Release 2: A Performance Analysis

January 2003

Author: Meikel Poess

Contributing Authors: Hermann Baer, Dmitry Potapov, Saroj Sancheti, Ray Glasstone, George Lumpkin

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
www.oracle.com

Oracle Corporation provides the software
that powers the internet.

Oracle is a registered trademark of Oracle Corporation. Various
product and service names referenced herein may be trademarks
of Oracle Corporation. All other product and service names
mentioned may be trademarks of their respective owners.

Copyright © 2000 Oracle Corporation
All rights reserved.